METISS Team

The Matching Pursuit Tool Kit

User Manual

Authors:
Sacha Krstulović
Rémi Gribonval

Contributors:
Ronan Le Boulch
Boris Mailhé
Benjamin Roy
Gilles Gonon

MPTK version : "0.6.0"
Document version : "1.0"

November 9, 2011

Contents

Part I

Getting Started

Abstract

This part of the document describes how to begin with Matching Pursuit Tool Kit. It provides informations about how to simply download, install and configure this library on every architecture from a released binary file. It also describes, in the case that the user have installed Matlab, the basic functions available and their usage.

1. Introduction

1.1 Learning about MPTK

The Matching Pursuit Tool Kit (MPTK) provides a fast implementation of the Matching Pursuit algorithm for the sparse decomposition of multichannel signals such as audio signals. It comprises a library, standalone utilities and Matlab scripts.

MPTK provides an implementation of Matching Pursuit which is:

- **FAST:** e.g., extract 1.5 million atoms from a 1 hour long, 16kHz audio signal (15dB extracted) in 0.25x real time on a Pentium IV@2.4GHz, out of a dictionary of 178M Gabor atoms. Such incredible speed makes it possible to process "real world" signals.
- **FLEXIBLE:** multi-channel, various signal input formats, flexible syntax to describe the dictionaries ↦ reproducible results, cleaner experiments.
- OPEN: modular architecture ↦ add your own atoms ! Free distribution under the GPL.

Signal	Dictionary		10dB SNR	20dB SNR	30dB SNR
Radio broadcast 1h @ 8kHz	dic_speech_8k	Short	nAtoms : 1.18 M Time : 1min	nAtoms : 4.75 M Time : 4min	nAtoms : 10.61 M Time : 10min
		Long	nAtoms : 0.98 M Time : 6min	nAtoms : 5.49 M Time : 23min	nAtoms : 5.49 M Time : 167min
Radio broadcast 1h @ 16kHz	dic_speech_16k	Short	nAtoms : 1.32 M Time : 3min	nAtoms : 6.08 M Time : 5min	nAtoms : 11,75 M Time : 15min
		Long	nAtoms : 1.28 M Time : 11min	nAtoms : 7.82 M Time : 73min	nAtoms : 14.14 M Time : 212min
Music song 3min @ 44kHz	dic_music_44k	Short	nAtoms : 81 K Time : 1min	nAtoms : 544 K Time : 5min	nAtoms : 1,81 M Time : 13min
		Long	nAtoms : 69 K Time : 5min	nAtoms : 735 K Time : 54min	nAtoms : 2.77 M Time : 201min

MPTK is mainly developed and maintained within the METISS Research Group (http://www.irisa.fr/metiss/) on audio signal processing, at the INRIA Research Institute (http://www.irisa.fr or http://www.inria.fr/rennes) in Rennes, France.

1.2 Reading this document

This document describes the basic principles about how to download, install and use the Matching Pursuit Tool Kit. It is divided into two major sections :

- **Part I : Getting started with MPTK**
  For Windows users : Read chapter 2 to learn how to download & install MPTK
  For Linux users : Read chapter 3 to learn how to download & install MPTK
  For Mac users : Read chapter 4 to learn how to download & install MPTK
  ↦ You can skip to chapters 5 & 6 to understand the basic usage of MPTK
- **Part II : Advanced User Manual**
  Read chapter 9 to learn how to pre-build, build & install MPTK from the source files
  Read chapters 11 & 12 to learn how to use MPTK with command line tools
  Read chapters 13 & 14 to learn how to use MPTK within Matlab

2. MPTK for Windows

2.1 Downloading MPTK

The latest version of MPTK is available at (https://gforge.inria.fr/frs/?group_id=36). Depending on the processor architecture of your computer, you will have to download either the 32 bits package or the 64 bits package:

- For Windows 32 bits : "MPTK-binary-v.v.v-i386-Windows.exe"
- For Windows 64 bits : "MPTK-binary-v.v.v-x86_64-Windows.exe"

Hint : To find the processor architecture of your computer:

- *Open a terminal command using :*
  - *Start ↦ All Programs ↦ Accessories ↦ Command Prompt*
- *Use the following command : echo %PROCESSOR_ARCHITECTURE%*
  - *If the answer is "x86" then your OS is 32 bits*
  - *If the answer is "AMD64" then your OS is 64 bits*

2.2 Installing MPTK

When double clicking the executable "MPTK-binary-v.v.v.-(i386/x86_64)-Windows.exe":

1. Accept the terms of the licence agreement
2. Select the path folder where to install MPTK (we'll call it *"path_to_MPTK"*)
   - We suggest to use the default folder : "C:\Program Files\MPTK"
3. Finish the installation

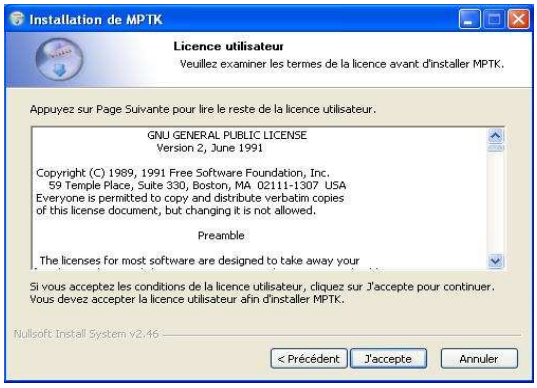

Figure 2.1: Licence agreement

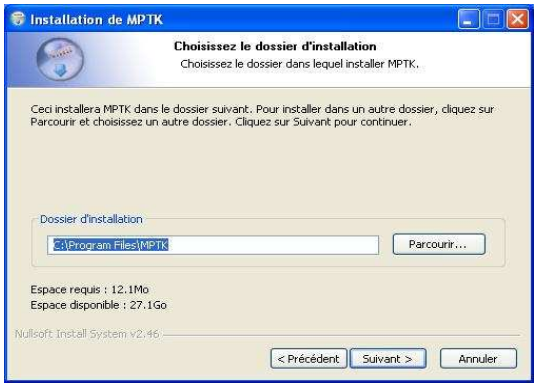

Figure 2.2: Path folder selection

2.3 Configuring the path

An environment variable called MPTK_CONFIG_FILENAME needs to be set, either temporarily, or permanently, with the path of the "path.xml" file located in the *"path_to_MPTK/mptk"* directory. This file defines the environment paths that MPTK needs to work correctly.

2.3.1 Temporary path configuration

Here is the way to temporarily configure the MPTK_CONFIG_FILENAME environment variable.
Warning : This is a temporary setting and it needs to be done at each reset of the computer.

- Open a terminal command using :
  - Start ↦ All Programs ↦ Accessories ↦ Command Prompt
- Use the command : *set MPTK_CONFIG_FILENAME = path_to_MPTK/mptk/path.xml*

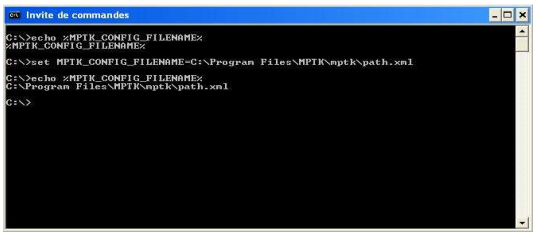

Figure 2.3: Filled command prompt

2.3.2 Permanent path configuration

Here is the way to permanently configure the MPTK_CONFIG_FILENAME environment variable

- Check if the environment variable is correctly set with : echo %MPTK_CONFIG_FILENAME%
- Open the environment variable configuration panel situated under :
  - Start ↦ Config panel ↦ System ↦ Advanced ↦ Environment variables
- Add a new user variable with :
  - Name : MPTK_CONFIG_FILENAME
  - Value : *path_to_MPTK/mptk/path.xml*

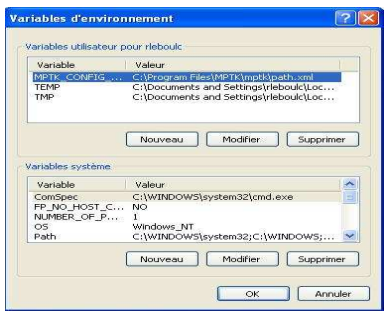

Figure 2.4: Environment variable configuration

2.3.3 Matlab path configuration

When launching Matlab, the user needs to configure Matlab to work with MPTK:

- Configure the working path either by:
  - Selecting the current folder as "path_to_MPTK/mptk/matlab"
  - Adding the working path using addpath("path_to_MPTK/mptk/matlab")

3. MPTK for Linux

3.1 Downloading MPTK

The latest version of MPTK is available at (https://gforge.inria.fr/frs/?group_id=36). Depending on the processor architecture of your computer, download either the 32 bits package or the 64 bits package:

- For RedHat, Suse, Fedora, Mandriva
  - 32 bits : "MPTK-binary-v.v.v-i386-Linux.rpm"
  - 64 bits : "MPTK-binary-v.v.v-x86_64-Linux.rpm"
- For Debian, Knoppix, Ubuntu
  - 32 bits : "MPTK-binary-v.v.v-i386-Linux.deb"
  - 64 bits : "MPTK-binary-v.v.v-x86_64-Linux.deb"

Hint : To find the processor architecture of your computer :

- *Open a terminal command and use the following command :* ***uname -m***
  - *If the answer is "i386" then your OS is 32 bits*
  - *If the answer is "x86_64" then your OS is 64 bits*

3.2 Obtaining additional required packages

Two additional packages are needed. Their installation require administrator privileges on the machine. The "sudo" command may ask you to input administrator password :

- Libsndfile (tested with version 1.0.23) pre compiled library
- FFTW (tested with version 3.2.2) pre compiled library

You can see below some examples about how to download those libraries using terminal :

Ubuntu	*Fedora*	*Mandriva*
sudo apt-get install -y -qq libsndfile1-dev *sudo apt-get install -y -qq libfftw3-dev*	*sudo yum -y -qq install libsndfile-devel* *sudo yum -y -qq install fftw-devel* *sudo yum -y -qq install fftw-static*	*sudo smart install -y -qq libsndfile1-dev* *sudo smart install -y -qq libfftw3-dev*

3.3 Installing MPTK

Depending on the type of Linux you have there ares two ways to install the packages :

- for "rpm" package : *rpm -ivh MPTK-binary-v.v.v-(i386/x86_64)-Linux.rpm*
- for "deb" package : *dpkg -i MPTK-binary-v.v.v-(i386/x86_64)-Linux.deb*

3.4 Configuring the path

An environment variable called MPTK_CONFIG_FILENAME needs to be set, either temporarily, either permanently, with the path of the "path.xml" file located in the *"path_to_MPTK/mptk"* directory. This file defines the environment paths that MPTK needs to work properly.

3.4.1 Temporary path configuration

Here is the way to temporarily configure the MPTK_CONFIG_FILENAME environment variable.
Warning : This is a temporary setting and it needs to be done at each reset of the computer.

- *With Bash shell :*
  - *export MPTK_CONFIG_FILENAME ="path_to_MPTK/mptk/path.xml"*
- *With C-shell :*
  - *setenv MPTK_CONFIG_FILENAME "path_to_MPTK/mptk/path.xml"*
- *You can check if the environment variable is correctly set with :*
  - *echo $MPTK_CONFIG_FILENAME*

3.4.2 Permanent path configuration

In order to permanently configure the MPTK_CONFIG_FILENAME environment variable, add the bash shell (or the C-shell) configuration line to the ".bashrc" (or the ".cshrc") file situated under your home directory.

3.4.3 Matlab path configuration

When launching Matlab, the user needs to configure Matlab to work with MPTK:

- Configure the working path either by:
  - Selecting the current folder as *"path_to_MPTK/mptk/matlab"*
  - Adding the working path using *addpath("path_to_MPTK/mptk/matlab")*

4. MPTK for Mac OS

4.1 Downloading MPTK

The latest version of MPTK is available at (https://gforge.inria.fr/frs/?group_id=36). Depending on the processor architecture of your computer, you will have to download either the 32 bits package or the 64 bits package:

- For Mac 32 bits : "MPTK-binary-v.v.v-i386-Mac.exe"
- For Mac 64 bits : 'MPTK-binary-v.v.v-x86_64-Mac.exe"

Hint : To find the processor architecture of your computer :

- *Open a terminal command and use the following command :* ***uname -m***
  - *If the answer is "i386" then your OS is 32 bits*
  - *If the answer is "x86_64" then your OS is 64 bits*

4.2 Obtaining additional required packages

Two additional packages are needed. Their installation require administrator privileges on the machine. The "sudo" command may ask you to input administrator password :

- Libsndfile (tested with version 1.0.23) pre compiled library
- FFTW (tested with version 3.2.2) pre compiled library

You can see below some examples about how to download those libraries using terminal:

Mac
sudo /opt/local/bin/port install libsndfile +universal
sudo /opt/local/bin/port install fftw-3 +universal

Hint : We suggest to use the "port" command from MacPorts because the command "+universal" allows to retrieve libraries which are compatible with both system architectures (32 bits and 64 bits). The package is available at http://www.macports.org/install.php

4.3 Installing MPTK

When double clicking the executable **"MPTK-binary-v.v.v.-(i386/x86_64)-Mac.dmg"**:

1. Accept the terms of the licence agreements
2. Accept the path folder where to install MPTK
   - The default and unique folder is : "/usr/local/"
3. Finish the installation

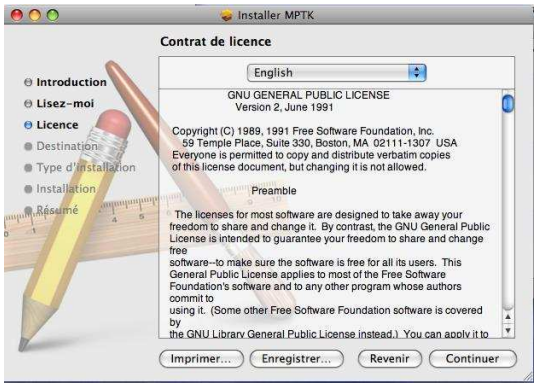

Figure 4.1: Licence agreement

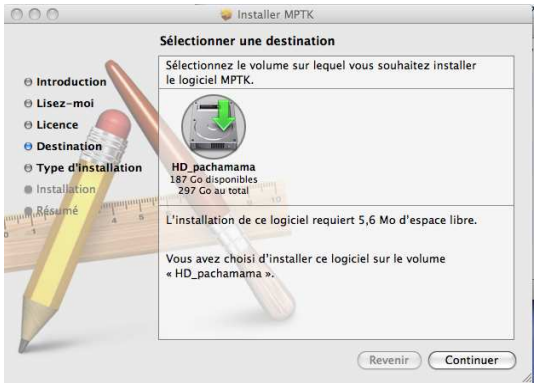

Figure 4.2: Path folder selection

4.4 Configuring the path

An environment variable called MPTK_CONFIG_FILENAME needs to be set, either temporary, either permanently, with the path of the path.xml file located in the *"path_to_MPTK/mptk"* directory. This file defines the environment paths that MPTK needs to work properly.

4.4.1 Temporary path configuration

Here is the way to temporary configure the MPTK_CONFIG_FILENAME environment variable. **Warning :** This is a temporary setting and it needs to be done at each reset of the computer.

- *With Bash shell :*
  - export MPTK_CONFIG_FILENAME ="/usr/local/mptk/path.xml"
- *With C-shell :*
  - setenv MPTK_CONFIG_FILENAME "/usr/local/mptk/path.xml"
- *You can check if the environment variable is correctly set with :*
  - echo $MPTK_CONFIG_FILENAME

4.4.2 Permanent path configuration

In order to permanently configure the MPTK_CONFIG_FILENAME environment variable, add the bash shell (or the C-shell) configuration sentence to the ".bashrc" (or the ".cshrc") file situated under your home directory.

4.4.3 Matlab path configuration

When launching Matlab, the user needs to configure Matlab to work with MPTK:

- Configure the working path either by :
  - Selecting the current folder as "/usr/local/mptk/matlab"
  - Adding the working path using addpath("/usr/local/mptk/matlab")

5. MPTK from within Matlab

In the following, we assume that MPTK has been installed correctly, that the system path configuration has been set and that Matlab is installed.

5.1 Getting Started

GettingStarted *command is used for a better understanding of MPTK functionalities. This script is divided in two parts. The first part consists in retrieving the environment informations and the available plugins. The second part consists in describing several tutorials about MPTK utilities.*

1st part description : Example of available plugins and dictionaries
Here is the list of types of atoms available in MPTK plugins:

- anywave	- anywavehilbert
- constant	- dirac
- gabor	- harmonic
- mclt	- mdct
- mdst	- nyquist

As well as information on the path where reference files can be found:

- *path_to_MPTK/mptk/reference*

which can be used to find examples of dictionaries:

dic_anywave.xml	dic_constant.xml	dic_harmonic.xml
dic_mdst.xml	dic_anywave_modifie.xml	dic_dirac.xml
dic_mclt.xml	dic_nyquist.xml	dic_chirp.xml
dic_gabor_two_scales.xml	dic_mdct_two_scales.xml	dic_test.xml

2nd part description : Description of the tutorials
There are several tutorials on using MPTK4Matlab:

1. Dictionaries
2. Books
3. Running Matching Pursuit Toolkit
4. Multichannel decompositions (in preparation)
5. Anywave atoms (in preparation)
6. Demixing pursuit (in preparation)

Dictionaries : How to read (*dictread*), create (*dictwrite*) a dictionary description
Books : What is a book (storage format for sparse signal representations) and how to read (*bookread*), save (*bookwrite*) or plot any book (*bookplot*, *bookover*)
Running MPTK : Procedure to follow if you want to decompose a signal :

- Read a signal (*sigread*)
- Read a dictionary (*dictread*)
- Decompose the signal (*mpdecomp*)

5.2 Getting the environment information

getmptkinfo *command is launched under the GettingStarted.m script, and is used to retrieve the environment datas, such as :*

- The plugins atoms names available
- The block parameters needed to correctly define each atom
- The windows names that can be used to through the signal waveform
- The example or default environment paths

↦ mptkInfo = getmptkinfo();

5.3 Reading a signal

sigread *command reads an imports signal "exampleSignal" of any format supported by libsndfile library to Matlab and gives a matrix "signal" (numSamples x numChans) and the sampling frequency of the read signal "sampleRate".*

↦ [signal sampleRate] = sigread(mptkInfo.path.exampleSignal);

5.4 Reading a dictionary

dictread *command imports a dictionary description "defaultDict" to Matlab and gives a dictionary description with the following structure : dict.block{i} = block where, for example block.type = 'dirac'*

↦ dict = dictread(mptkInfo.path.defaultDict);

5.5 Decomposing a signal

mpdecomp *command decompose a signal "Signal" using its sampling frequency "sampleRate", a dictionary structure "dict" performing "numIter" iterations and gives the resulting decomposition "book", the "residual" obtained after the iterations and "decay", a vector with the energy of the residual after each iteration.*

↦ [book residual decay] = mpdecomp(signal,sampleRate,dict,numIter);

5.6 Plotting a book

bookover *plots the given "book" over a STFT spectrogram of the given "Signal" for channel "numChan" (or 1 for default). The book and/or the signal can be given as filenames (WAV format for the signal).*

↦ figure(5);bookover(book,signal);

5.7 Reconstructing a signal

mprecons *reconstructs the signal from the given "book".*

↦ sigrec = mprecons(book);

5.8 Finding further informations on MPTK for Matlab

For further informations on how to use MPTK Matlab functionalities please refer to the Chapter 13 of the Usermanual situated under the under the documentation directory (MPTK UserManual) or under the gforge website (https://gforge.inria.fr/docman/?group_id=36).

6. MPTK command line utilities

6.1 List of commands

Here is a list of the MPTK command line utilities :

- **mpd :** decompose a waveform signal using matching pursuit
- **gpd :** decompose a waveform signal using gradient pursuit
- **mpd_demix :** decompose a waveform signalusing matching pursuit and a mixer matrix
- **mpf :** filters the atoms contained in books
- **mpr :** reconstructs a signal from the atoms contained in a book
- **mpcat :** concatenates any number of books into one
- **mpview :** makes a time-frequency pixmap

6.2 Basic example

Here is a simple example on how to use MPTK with terminal commands :

---

mpd command iterates Matching Pursuit on signal "sndFileToDecomp.wav" with dictionary "dictFile.xml" and gives the resulting book "bookFile.bin" (and an optional residual signal) after N iterations or after reaching the signal-to-residual ratio SNR.

```
$ mpd -s 10 -R 10 -d dictFile.xml sndFileToDecomp.wav bookFile.bin
```

---

mpr commands rebuilds a signal "sndReconsFile.wav" from the atoms contained in the book file "bookFile.bin". An optional residual "sndResidFile.wav" can be added.

```
$ mpr bookFile.bin sndReconsFile.wav sndResidFile.wav
```

information : Some book examples are available under "path_to_MPTK/mptk/reference/book"

---

mpf commands filters the atoms contained in "bookFile.bin" (or stdin), stores those which satisfy the indicated properties in "bookYes.bin" (or stdout) and the others in "bookNo.bin".

```
$ mpf –Freq=[0:110] –len=[0:256] bookFile.bin bookYes.bin bookNo.bin
```

6.3 Finding further informations on MPTK cmd line

For further informations on how to use MPTK command line utilities please refer to the Chapter 11 of the Usermanual situated under the under the documentation directory (MPTK UserManual) or under the gforge website (https://gforge.inria.fr/docman/?group_id=36).

7. Help, contact and forums

If you need help with the software:

1. Check if a more recent release fixes your problem (https://gforge.inria.fr/frs/?group_id=36)
2. Check if somebody else has had a samilar problem and if a fix exist on the help forum (https://gforge.inria.fr/forum/forum.php?forum_id=109)
3. If not, post a message on the help forum

If you need documentation about the software:

Some articles exposing scientific results related to MPTK are available in PDF format through the following page (https://gforge.inria.fr/docman/?group_id=36)

If you want specific information :

You can write to us to matching.pursuit@irisa.fr.

Request for help sent to this address won't be answered.
Please use the Help forum instead.

Thank you for your interest in The Matching Pursuit ToolKit !

Part II

User Manual

Abstract

This part of the document describes the basic principles of the Matching Pursuit Tool Kit, and some basic guidelines about how to use it. The appendix also provides the potential contributor with notes about the insides of our implementation, plus guidelines about some of the development conventions that we have adopted. For a more detailed documentation of the code, see the automatically generated reference manual.

8. Basic understanding

8.1 Terminology

Atom – An elementary piece of signal. An atom is characterized by its class (e.g., Gabor atoms, harmonic atoms etc.) and a set of parameters (e.g., for a Gabor atom, its time location and length, its frequency, its amplitude, a chirp factor and a given shaping window). In our code, the corresponding object knows how to subtract itself from a signal, and how to generate its own waveform.

Block – A block computes the correlations (inner products) between a signal to analyze and a set of atoms for which the computation of the correlations can be factorized in an efficient manner, along the whole support of a signal. For instance, the inner products can stem from a time-frequency transform, such as the Fourier Transform or a wavelet transform, provided the transform can be interpreted as a set of correlations between the signal and a set of atoms which cover the time-frequency plane.

Each block object can search for the location of the maximum correlation between the atoms and the signal, and can deduce which atom contributes the most energy to the analyzed signal. Several blocks corresponding to various scales or various transforms can be concurrently applied to the same signal, thus providing for multi-scale or multiple-basis analysis.

The following blocks are implemented at the present time :

- **Gabor** : Based on Short-Time Fourier Transforms, which compute the correlations with windowed sinusoids having a "flat" frequency (chirp rate == 0). In this case, one block is conceptually equivalent to one STFT with a given scale.
- **Harmonic** : Based on the grouping of fundamental and multiples Gabor atoms.
- **Chirp** : Based on the Gabor atom with (linear) time variable frequency.
- **AnyWave** : Based on a fast convolution algorithm to compute some correlations with arbitrary waveforms.
- **MDCT/MDST/MCLT** : Transforms based on local cosine functions.
- **Dirac** : Find the signal samples which have a high energy
- **Constant** : Rectangular windows, defined by a length and a shift between atoms. A constant atom catches the mean of a signal on the specified frame.
- **Nyquist** : Defined by a length and a shift between atoms, and the waveform is a normalized succession of 1 and -1. A Nyquist atom catches the greatest distinguished frequency (called the Nyquist frequency if the support is even) of the specified frame.

In the future, blocks based on fast wavelet transforms should also be designed.

Dictionary – A dictionary contains a collection of blocks plus the signal on which they operate. It can search across all the blocks (i.e., all the scales and all the bases) for the atom which brings the most energy to the analyzed signal.

Book – A book is a collection of atoms and its corresponding dictionaries. Summing all the atoms in a book gives a signal. The *Figure 8.1* illustrates the fact that a dictionary

contains a signal and a collection of blocks, and that a books contains a collection of atoms. The algorithm linking these objects will be explained in the next section.

8.2 Algorithm

Our implementation of the Matching Pursuit algorithm uses roughly 3 steps, as illustrated in figure 8.1:

1. Update the correlations in the blocks, by applying the relevant correlation computation algorithm to the analyzed signal, and find the maximum correlation in the same loop
2. Create the atom which corresponds to the maximum correlation with the signal (and store this atom in the book)
3. Subtract the created atom from the analyzed signal, thus obtaining a residual signal, and re-iterate the analysis on this residual

At each step, the original signal can be rebuilt exactly by summing all the atoms of the book and adding them to the residual signal.

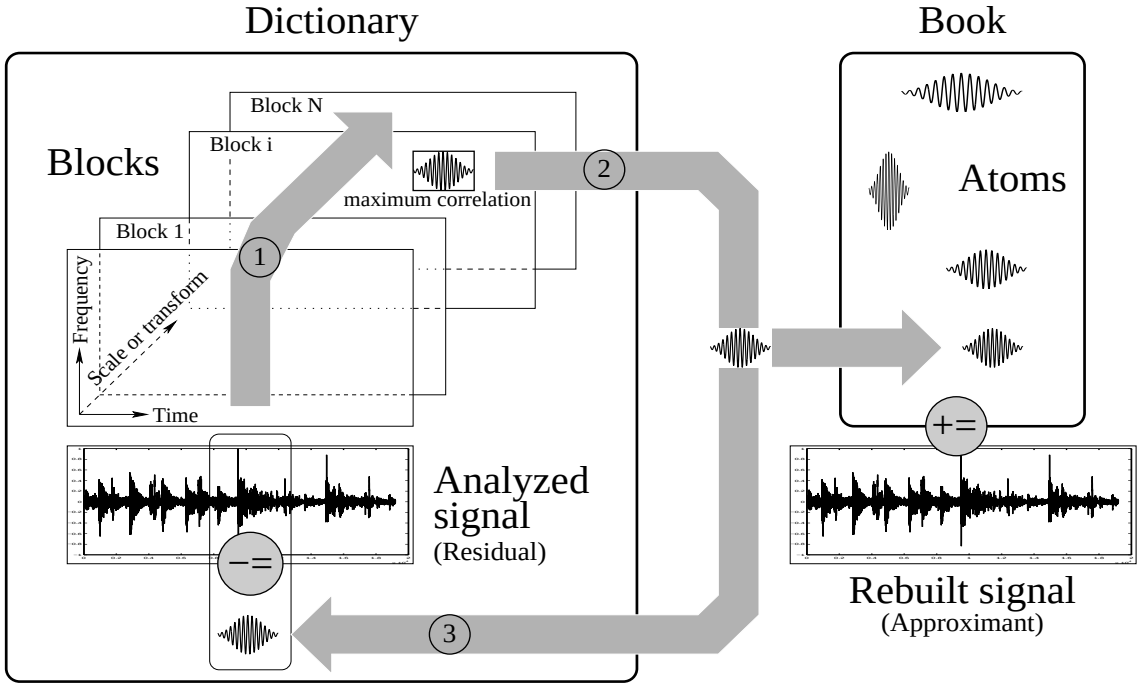

Figure 8.1: Description of Matching Pursuit. (See sections 8.1/8.2 for more explanations)

9. Basic download & install from source

9.1 Getting the sources

The MPTK sources archives can be found under the gforge website, where you will be able to download any version of MPTK (https://gforge.inria.fr/frs/?group_id=36) :

- MPTK source archives : "MPTK-Source-v.v.v.tar.gz" (current version is 0.6.0)

9.2 Basic installation under Windows

9.2.1 Required tools

- Necessary :
  - Cmake (tested with version 2.8.3)
    $\mapsto$ *To obtain it : http://www.cmake.org/cmake/resources/software.html*
  - Microsoft Visual Studio (tested with free Visual C++ 2008 Express)
    $\mapsto$ *To obtain it : http://www.microsoft.com/express*
- Optional (if Matlab is needed) :
  - Matlab at least version R2006, **preferably** installed in the default directory :
    * C:\Program Files\MATLAB

If the required tools are installed as mentioned, you should be able to generate a build system using the basic build settings.

9.2.2 Pre-configuration using cmake

Suppose you unzip MPTK-Source-v.v.v.tar.gz in directory c:\foo and you want to build MPTK files in c:\bar (you may choose to build in a temporary location which differ from the place where you will eventually install MPTK). Use CMake to generate the build system, use the Make icons from the program menu to launch the CMake configuration application :

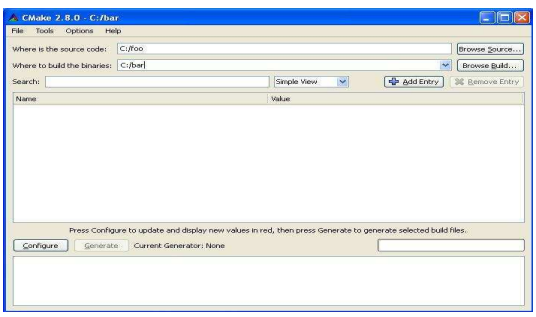

Figure 9.1: Cmake view under windows

Set the “Where is the source code” text box with the path of the directory where the source files are located (c:\foo) and the “Where to build the binaries” with the path of the directory where you want to build the library and executable files (c:\bar).

When clicking for the first time on the [Configure] button, CMake will ask for the build tool you want to use. The build system type depends on the builder you want to use, in our case this is the Visual Studio 9 chain tools :

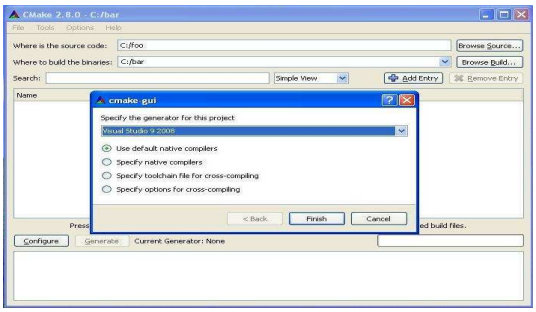

Figure 9.2: Cmake configuration using Visual Studio

When pressing again the [Configure] button to configure the build system, CMake performs a list of tests to determine the system configuration and manage the build system. If the configuration is correct then no pop-up will appears during the tests and CMake finally shows the various options of the build underlaid in grey. In case of a configuration issue, a pop up window warns you about this issue indicating which test has failed, in this case the build option in the CMake application software will be underlaid in red. We will discuss in section 10.4 “Installation Troubleshooting” what to do in such a case, but let us for the moment assume that everything ran smoothly.

Once the build system is configured then generated, you have to build MPTK using Visual.

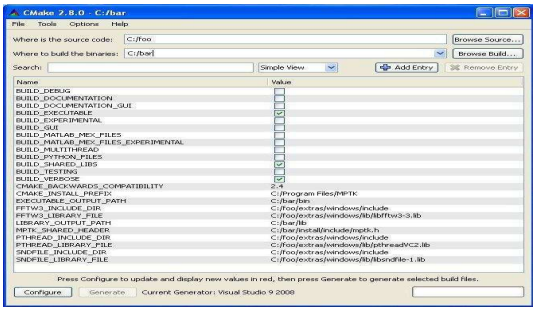

Figure 9.3: Cmake build system view for MPTK

9.2.3 Basic build

Once the project is generated using Cmake, the build directory (ex : C:/bar) contains a Visual solution (MPTK.sln). When you double click on this file, Visual Studio software starts and opens the differents projects associated with this solution.

Right clicking on the project "ALL_BUILD" located on the left tab of Visual Studio, then selecting "Generate" will compile all the MPTK binary files.

9.2.4 Basic installation

Right clicking on the project "INSTALL" located on the left tab of Visual Studio, then selecting "This Project" ↦ "Generate only INSTALL" will install MPTK to the directory associated with the "MPTK_INSTALL_PREFIX" variable of Cmake.

9.3 Basic installation under Unix & MacOS

If the required libraries are installed as the appendix A and B indicate it, you should be able to generate a build system using the basic build settings.

9.3.1 Required tools

- Necessary :
  - Cmake (tested with version 2.8.3)
    ↦ *To obtain it : http://www.cmake.org/cmake/resources/software.html*
- Optional (if Matlab is needed) :
  - Matlab at least version R2006 (**Caution** install Matlab in the default directory) :
    - For Linux : /opt/MATLAB_Ryyyyx or /soft/MATLAB_Ryyyyx
    - For Mac : /Applications/MATLAB_Ryyyyx (with yyyy = year, x=[a-z])

9.3.2 Required packages

- Libsndfile pre-compiled library (tested with version 1.0.21)
- FFTW pre-compiled library (tested with version 3.2.2)

9.3.3 Pre-configuration using cmake

Suppose you untar "MPTK-Source-v.v.v.tar.gz" in directory ~/foo and you want to build MPTK binary and library files in ~/bar.

The Command list you should use will look like this :

```
bash-3.1$ mkdir ~/bar
bash-3.1$ cd ~/bar
bash-3.1$ cmake ~/foo
```

When using the cmake command to generate the build system, Cmake performs a list of tests to determine the system configuration and manage the build system. If the configuration is correct then the build system is generated and written. In this case the three last lines of the console log of cmake command should be:

```
...
bash-3.1$ -- Configuring done
bash-3.1$ -- Generating done
bash-3.1$ -- Build files have been written to:  ~/bar
```

9.3.4 Basic build

The command `make` will compile the build files and should finish like :

```
...
bash-3.1$ Linking CXX executable ../../bin/test_windowLen
bash-3.1$ [100%] Built target test_windowLen
```

9.3.5 Basic installation

The command `sudo make install` will install in default directory "path_to_MPTK"and should finish like :

```
...
bash-3.1$ -- Installing:  /usr/local/lib/liblibmd5sum.a
bash-3.1$ -- Installing:  /usr/local/lib/liblibgetopt.a
bash-3.1$ -- Installing:  /usr/local/doc/short_description.pdf
bash-3.1$ -- Installing:  /usr/local/doc/GettingStarted.pdf
bash-3.1$ -- Installing:  /usr/local/doc/userman.pdf
```

10. CMake options & troubleshoot

10.1 Using ccmake to set build options

The Cmake Graphical User Interface is automaticaly displayed under Windows OS, and is available with the command `ccmake` under Unix OS. It allows you to :

- Specify build system options, such as either to enable the use of multiple threads to speed up the Matching Pursuit decomposition or to generate the doxygen documentation
- Troubleshoot the various paths required to build MPTK
- Display advanced build options via the advanced mode

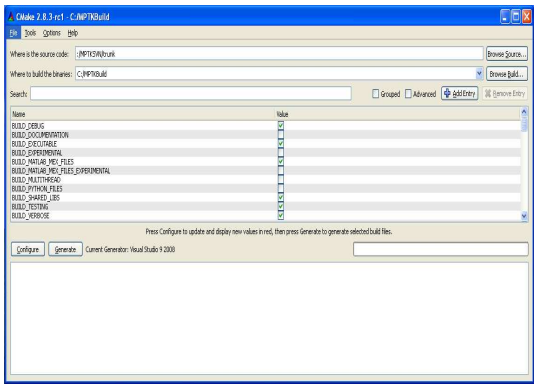

Figure 10.1: Windows CMake GUI

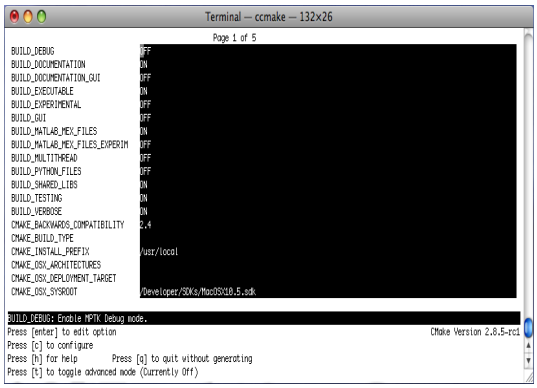

Figure 10.2: Unix CMake GUI

10.2 Main options

This Graphical User Interface allows the user to set the build options for MPTK :

- CMAKE_INSTALL_PREFIX : Path of the installation (by default: /usr/local/)
- BUILD_DOCUMENTATION : Generation of the doxygen documentation library. This operation can take a lot of time, be patient. (by default : OFF)
- BUILD_DOCUMENTATION_GUI : Generation of the doxygen documentation for the Graphical User Interface. Careful, it can take a lot of time. (by default : OFF)
- MULTITHREAD : Allow MPTK to use parrallel computing, this option allows to increase performance on multi-processor hardware architectures (by default : OFF)

10.3 Advanced options

You can display more information and all the configuration options by switching to the advanced mode, which is set to off by defaut and can be toggled on by pressing [t].

The list of the following options should be considered as advanced:

- BUILD_DEBUG : Compile in debug mode (by default: OFF = Release)
- BUILD_SHARED_LIBS: Allows the construction of libraries (libdsp_windows & libmptk) in a lib directory and the include file (mptk.h) in an include directory (by default: ON)
- BUILD_TESTING: Allow to test executables using Dart server (by default : ON)
- BUILD_VERBOSE: Display various informations during the build (by default : ON)
- BUILD_EXECUTABLE: Build the executables mpd, mpf... (by default: ON)

10.4 Troubleshooting build issues

10.4.1 CMake GUI issues

Let's see a more complex use of the Cmake GUI troubleshooting. Despite all the integrated tests when Cmake manages to generate the build system of MPTK, sometimes Cmake cannot determine some parameters of your configuration. When using the cmake command, Cmake performs a list of tests on the build system configuration in order to generate it, if there is some issue related to the configuration, the errors list is displayed at the end of the consol log. Here we have 4 includes files or libraries that Cmake cannot found :

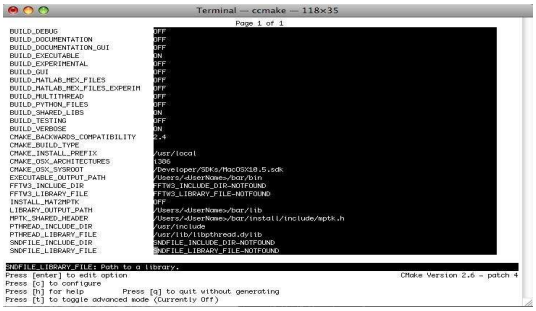

Figure 10.3: Example view of cmake libraries detection failed

In this current case, `FFTW3_INCLUDE_DIR`, `FFTW3_LIBRARY_FILE`, `SNDFILE_INCLUDE_DIR` and `SND_LIBRARY_FILE` fields must be set with the path to the directory containing the FFTW and sndfile libraries. To set the fields values simply press [enter] while the cursor is on the relevant field, and after editing press [enter] to escape the field edition.

Important note: In the case of include files, only the directory where this include files are located is required. But in the case of library files, the entire path to the library including its name must be indicated in the corresponding field.

After choosing the options for the build and setting the required fields, press [c] to configure. The configuration of the build system is checked again by Cmake, at the end of this check if the build settings are correct, you can press [g] in order to generate the build system and quit the Cmake GUI.

10.4.2 Fixing FFTW used plan with wisdom in FFTW configuration

MPTK currently relies on FFTW to compute Fast Fourier Transforms. The principle of FFTW is to adaptively select the fastest running codelet to perform a given FFT, which is good for performance. However, this can lead to non deterministic behaviour of MPTK since two consecutive runs of the same command (e.g. 'mpd') with the same options on the same data can yield slightly different results. To avoid this issue we have implemented a mechanism which allows MPTK to fix which FFT codelet is used for a specific type of FFT computation. To do this, you just need to create and export a environment variable called MPTK_FFTW_WISDOM and set this environnement variable with the path of the file you want to use in order to save the FFTW configuration. MPTK will create a "wisdom" file, that will be reloaded each time you use MPTK. It means that the codelets use to compute FFTW plan will be the same between two successive decompositions with the same parameters.

For bash or sh or ash:

```
bash-3.1$ export MPTK_FFTW_WISDOM=/usr/local/fftw_wisdom_file
```

If you want to fix permanently the used FFTW plan, just set this environment variable when the console is launched, using for that the /etc/profile, /.profile or .bashrc file, according to your system configuration.

For tcsh or csh :

```
bash-3.1$ setenv MPTK_FFTW_WISDOM=/usr/local/fftw_wisdom_file
```

If you want to fix permanently the used FFTW plan, just set this environment variable when the console is launched, using for that the /etc/csh.cshrc file, according to your system configuration.

11. How to use MPTK command lines

11.1 Beginning with command lines

Unix & MacOS:

The installation is supposed to copy the binary files (mpd, mpr, mpf...) under "/usr/local/bin" and the library files under "/usr/local/lib" which are supposed to be under the environment PATH destination. You should normaly be able to launch it directly using terminal command.

Try typing "mpd" under a terminal command and then you should see the help description.

Windows:

The installation is supposed to copy the binary files (mpd, mpr, mpf...) and the library files under "C:\Program Files\MPTK\bin". In order to make it work, type "cmd" under the execute launcher. Then try typing "C:\Program Files\MPTK\bin\mpd.exe" and then you should see the help description.

11.2 Introduction

The Matching Pursuit standalone utilities are designed so that a typical processing task decomposes into a chain of commands which can be connected by pipes. The signal decomposition is performed by `mpd`. The atoms stored in the resulting books can be filtered (i.e., sorted and selected) with the command `mpf`. Several books can then be concatenated with the command `mpcat`. Once a desired book is obtained, a corresponding approximant signal can be rebuilt using `mpr`, with the optional addition of a residual signal (or in addition to any other signal). Alternately, the `mpd_demix` utility provides blind source separation using Matching Pursuit when the mixing matrix is known. Each of the cited utilities have a `-h` switch to get command-line help.

Some typical processing chains include:

- Signal approximation with N atoms:
  ```
  mpd -D dictionary -n N soundFile.wav book.bin;
  mpr book.bin approx.wav
  ```
  – or –
  ```
  mpd -D dictionary -n N soundFile.wav - | mpr - approx.wav
  ```
- Exact reconstruction:
  ```
  mpd -D dictionary -n N soundFile.wav book.bin residual.wav;
  mpr book.bin approx.wav residual.wav
  ```
- Signal approximation, using only the short atoms (e.g., of less than 128 samples) and ignoring the residual:
  ```
  mpd -D dictionary -n N soundFile.wav book.bin;
  ```

```
mpf --length=[0:128] book.bin bookShort.bin;
mpr bookShort.bin approx.wav
```

- Blind source separation and rebuilding an approximation of one of the sources (e.g., the 3rd one):
```
mpd_demix -D dictionary -M matrix.txt -n N mix.wav bookFile;
mpr bookFile_3.bin approxOfSource3.wav
```

Lots of other combinations are possible. The relevant utilities are described with more detail in the next sections.

Note about the signal file formats: MPTK has been originally designed for sound processing, but it is applicable to any kind of signal.

For input, we have mostly been using the WAV format, but any "fees free" file format recognized by the `libsndfile` library (OGG, AIFF, FLAC...) should work with the provided MPTK utilities. MP3 is not supported for one very good reason, doing so requires the payment of licensing fees.

The signal output is more specific: the MPTK binaries output signals in the WAV format only. *(WARNING: in the `libsndfile` implementation, this format is not protected against clipping, which may happen for some books and is not an artifact of the MPTK analysis.)* Nevertheless, the MPTK API also offers Matlab, raw `double` and raw `float` signal output formats: those could be quickly enabled by simple hacks of the relevant utilities. To enable other output formats, please contribute some code to the `mp_signal.{h,cpp}` API.

11.3 The MPTK environment

In order to define the working environment of Matching Pursuit utilities, An XML file is loaded before using Matching Pursuit. This XML file contains the configuration path `<configpath>` path`</configpath>` the syntax of the path is `<path name="NAME" path="PATH"/>` the path used are:

- `dll_directory` the directory where MPTK library search for plugins Dynamic Link Library (DLL) defining atoms and blocks
- `fftw_wisdomfile` the default path for the FFTW wisdom file which allows to save the setting for FFT computation.

This values can be set by two way, the complete path of the file can be specified with the `-C<FILE>, -config-file=<FILE>` for utilities or MPTK library calls the environment variable `MPTK_CONFIG_FILENAME` to determine which file to use for setting up the MPTK environment.

12. List of command lines functions

12.1 mpd: matching pursuit signal decomposition

```
Usage:
    mpd [options] (-D|-d) dictFILE.xml -n N [-s SNR] (sndFILE.wav|-) (bookFILE(.bin|.xml)|-) [residualFILE.wav]

Synopsis:
    Iterates Matching Pursuit on signal sndFILE.wav with dictionary dictFILE.xml
    and gives the resulting book bookFILE (and an optional residual signal)
    after N iterations or after reaching the signal-to-residual ratio SNR.

Main arguments:
    (-D|-d) <dictFILE.xml>         The full/relative ("path_to_mptk/mptk/reference/dictionary/") dict path.
    -n<N>|--nIter=<N>|--nAtom=<N> Stop after N iterations or N atom founded.
    -s<SNR>, --snr=<SNR>           OPTIONAL: Stop when the SNR value <SNR> is reached.
    (sndFILE.wav|-)                The signal to analyze or stdin (in WAV format).
    (bookFILE.bin|bookFile.xml|-)  The file to store the resulting book of atoms, or stdout.
    residualFILE.wav               OPTIONAL: The residual signal after subtraction of the atoms.

Optional arguments:
    -C<FILE>, --config-file=<FILE>   Use the specified configuration file, otherwise MPTK_CONFIG_FILENAME
    -E<FILE>, --energy-decay=<FILE>  Save the energy decay as doubles to file FILE.
    -R<N>,    --report-hit=<N>       Report some progress info (in stderr) every N iterations.
    -S<N>,    --save-hit=<N>         Save the output files every N iterations.
    -T<N>,    --snr-hit=<N>          Test the SNR every N iterations only (instead of each iteration).
    -p<double>, --preemp=<double>    Pre-emphasize the input signal with coefficient <double>.
    -q, --quiet                      No text output.
    -v, --verbose                    Verbose.
    -V, --version                    Output the version and exit.
    -h, --help                       This help.

Examples:
    mpd -D /user/local/mptk/reference/dictionary/dic_gabor_two_scales.xml -n 10 glockenspiel.wav bookTest.bin
    mpd -d dic_gabor_two_scales.xml -n 10 -s 2.5 glockenspiel.wav bookTest.xml
```

12.2 gpd: gradient pursuit signal decomposition

```
Usage:
    gpd [options] -D dictFILE.xml -n N [-s SNR] (sndFILE.wav|-) (bookFILE.bin|-) [residualFILE.wav]

Synopsis:
    Iterates (Trunctated) Gradient Pursuit on signal sndFILE.wav with dictionary dictFILE.xml
     and gives the resulting book bookFILE.bin (and an optional residual signal)
    after N iterations or after reaching the signal-to-residual ratio SNR.

Main arguments:
     -D <dictFILE.xml>              Read the dictionary from xml dictFILE.
     -n<N>, --nIter=<N>|--nAtom=<N> Stop after N iterations or N atom founded.
     -s<SNR>, --snr=<SNR>           OPTIONAL: Stop when the SNR value <SNR> is reached.
     (sndFILE.wav|-)                The signal to analyze or stdin (in WAV format).
     (bookFILE.bin|bookFile.xml|-)  The file to store the resulting book of atoms, or stdout.
     residualFILE.wav               OPTIONAL: The residual signal after subtraction of the atoms.

Optional arguments:
    -C<FILE>, --config-file=<FILE>   Use the specified configuration file, otherwise MPTK_CONFIG_FILENAME
    -E<FILE>, --energy-decay=<FILE>  Save the energy decay as doubles to file FILE.
    -R<N>,    --report-hit=<N>       Report some progress info (in stderr) every N iterations.
    -S<N>,    --save-hit=<N>         Save the output files every N iterations.
    -T<N>,    --snr-hit=<N>          Test the SNR every N iterations only (instead of each iteration).
    -p<double>, --preemp=<double>    Pre-emphasize the input signal with coefficient <double>.
    -q, --quiet                      No text output.
    -v, --verbose                    Verbose.
    -V, --version                    Output the version and exit.
    -h, --help                       This help.

Examples:
    gpd -D /usr/local/mptk/reference/dictionary/dic_gabor_two_scales.xml -n 10 glockenspiel.wav bookTest.bin
```

12.3 mpr: signal reconstruction

```
Usage:
    mpr [options] (bookFILE.bin|bookFILE.xml|-) (reconsFILE.wav|-) [residualFILE.wav]

Synopsis:
    Rebuilds a signal reconsFile.wav from the atoms contained in the book bookFile.bin.
    An optional residual can be added.

Main arguments:
    (bookFILE.bin|bookFILE.xml-)     A book of atoms, or stdin.
    (reconsFILE.wav|-)               A file to store the rebuilt signal, or stdout (in WAV format).
    residualFILE.wav                 OPTIONAL: A residual signal obtained from a Matching Pursuit decomposition.

Optional arguments:
    -C<FILE>, --config-file=<FILE>   Use the specified configuration file, otherwise MPTK_CONFIG_FILENAME
    -d<double>, --deemp=<double>     De-emphasize the signal with coefficient <double>.
    -q, --quiet                      No text output.
    -v, --verbose                    Verbose.
    -V, --version                    Output the version and exit.
    -h, --help                       This help.
```

12.4 mpf: filtering of the book files

```
Usage:
    mpf --PROPERTY_1=[min:max] ... --PROPERTY_N=[min:max] (bookIn.bin|-) [bookYes.bin|-] [bookNo.bin]

Synopsis:
    Filters the atoms contained in bookIn.bin (or stdin), stores those which satisfy the
    indicated properties in bookYes.bin (or stdout) and the others in bookNo.bin (If no
    output files are given, the atoms are just counted and their number is reported in stderr).

Main arguments:
    (bookIn.bin|-)  A book of input atoms, or stdin.
    (bookYes.bin|-) OPTIONAL: A file (or stdout) to store the satisfying book of atoms.
    bookNo.bin      OPTIONAL: A file to store the non-satisying book of atoms.

Optional arguments:
    One or more of the following switches:
    --index=[min:max]    / -i [min:max] : keep the atoms ordered from min to max in the book
    --length=[min:max]   / -l [min:max] : keep a specific range of atom lengths (in number of samples)
    --Length=[min:max]   / -L [min:max] : keep a specific range of atom lengths (in seconds)
    --position=[min:max] / -p [min:max] : keep a specific range of atom positions (in number of samples)
    --Position=[min:max] / -P [min:max] : keep a specific range of atom positions (in seconds)
    --freq=[min:max]     / -f [min:max] : keep a specific frequency range (normalized between 0 and 0.5)
    --Freq=[min:max]     / -F [min:max] : keep a specific frequency range (Hz)
    --amp=[min:max]      / -a [min:max] : keep a specific range of amplitudes
    --chirp=[min:max]    / -c [min:max] : keep a specific range of chirp factors
    The intervals can exclude the min or max using reverted braces, e.g. ]min:max] exclude the min value.
    The intervals can be negated with prepending the '^' character, e.g. ^[min:max].

    --type=gabor|harmonic|dirac|anywave / -t gabor|harmonic|dirac|anywave : test the atom type.
    (The chirp type is not provided: a chirp atom is a gabor atom with a non-null chirp rate.)

Other optional arguments are:
    -C<FILE>, --config-file=<FILE>  Use the specified configuration file, otherwise MPTK_CONFIG_FILENAME
    -q, --quiet                     No text output.
    -v, --verbose                   Verbose.
    -V, --version                   Output the version and exit.
    -h, --help                      This help.

Example:
    mpf --index=[0:100[ --Freq=^[50:1000] bookIn.bin bookYes.bin bookNo.bin
    Take all the atoms with a frequency lower than 50Hz and higher than 1000Hz among the first 100
    atoms of bookIn.bin, store them in bookYes.bin and store all the others in bookNo.bin:

Note:
    Only one instance of each property is allowed. If you want to make more complicated domains, use a pipe.
```

12.5 mpcat: concatenation of book files

```
Usage:
    mpcat (book1.bin|-) (book2.bin|-) ... (bookN.bin|-) (bookOut.bin|-)

Synopsis:
    Concatenates the N books book1.bin...bookN.bin into the book file bookOut.bin.

Main arguments:
    (bookN.bin|-)        At least 2 books (or stdin) to concatenate.
    (bookOut.bin|-)      A book where to store the concatenated books, or stdout

Optional arguments:
    -C<FILE>, --config-file=<FILE>  Use the specified configuration file, otherwise MPTK_CONFIG_FILENAME
    -f, --force                     Force the overwriting of bookOut.bin.
    -q, --quiet                     No text output.
    -v, --verbose                   Verbose.
    -V, --version                   Output the version and exit.
    -h, --help                      This help.
```

12.6 mpd_demix: blind source separation with a matrix

```
Usage:
    mpd_demix [options] -D dictFILE.txt -M matrix.txt -n N [-s SNR] (sndFILE.wav|-) (bookFILE) [residualFILE.wav]

Synopsis:
    Performs Blind Source Separation on signal sndFILE.wav with dictionary dictFile.txt
    and with the known mixer matrix mixFILE.txt. The result is stored in as many books
    as estimated sources (plus an optional residual signal), after N iterations
    or after reaching the signal-to-residual ratio SNR.

Mandatory arguments:
    -M<FILE>, --mix-matrix=<FILE> Read the mixer matrix from text FILE. The first line of the file should indicate
                                  the number of rows and the number of columns, and the following lines should
                                  give space-separated values, with a line break after each row. Example :
                                         2 3
                                         0.92  0.38  0.71
                                         0.71  0.77  1.85
    -n<N>|--nIter=<N>|--nAtom=<N> Stop after N iterations or N atom founded.
    -s<SNR>, --snr=<SNR>          OPTIONAL: Stop when the SNR value <SNR> is reached.
    (sndFILE.wav|-)               The signal to analyze or stdin (in WAV format).
    (bookFILE)                    The base name of the files to store the books of atoms_n corresponding to the
                                  N estimated sources. These N books will be named bookFILE_n.bin, n=[0,...,N-1].

Optional arguments:
    -C<FILE>, --config-file=<FILE>   Use the specified configuration file, otherwise MPTK_CONFIG_FILENAME
    -E<FILE>, --energy-decay=<FILE>  Save the energy decay as doubles to file FILE.
    -R<N>,    --report-hit=<N>       Report some progress info (in stderr) every N iterations.
    -S<N>,    --save-hit=<N>         Save the output files every N iterations.
    -T<N>,    --snr-hit=<N>          Test the SNR every N iterations only (instead of each iteration).
    -p<double>, --preemp=<double>    Pre-emphasize the input signal with coefficient <double>.
    -q, --quiet                      No text output.
    -v, --verbose                    Verbose.
    -V, --version                    Output the version and exit.
    -h, --help                       This help.
```

12.7 mpview: generation of a time-frequency map from a book

```
Usage:
    mpview [options] (bookFILE.bin|-) tfmapFILE.flt

Synopsis:
    Makes a time-frequency pixmap fill it with the time-frequency representation
    of the atoms contained in the book file bookFile.bin and write it to the file
    tfmapFILE.flt as a raw sequence of floats.
    The pixmap size is 640x480 pixels unless option --size is used.
    Each time-frequency atom is displayed with a rectangle unless
    unless option --wigner is used to display its pseudo Wigner-Ville distribution

Mandatory arguments:
    (bookFILE.bin|-)     A book of atoms, or stdin.
    tfmapFILE.flt        The file where to write the pixmap in float.

Optional arguments:
    -C<FILE>, --config-file=<FILE>   Use the specified configuration file, otherwise MPTK_CONFIG_FILENAME
    -s, --size=<numCols>x<numRows>   Change the size of the pixmap.
    -w, --wigner Change the display mode.
    -q, --quiet No text output.
    -v, --verbose Verbose.
    -V, --version Output the version and exit.
    -h, --help This help.
```

13. How to use MPTK with Matlab

13.1 Beginning with Matlab

MATLAB (MAtrix LABoratory) is a numerical computing environment. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.

The first step to understand when using Matlab is the path configuration. In fact, it works as a terminal : The available files are the ones present in the current selected path. When launching MATLAB, the default path is not the one required. If the user wants to change it (in order to include MPTK path), there are three possibilities :

- Using the MATLAB native command "addpath(path_to_MPTK/mptk/matlab)"
- Using the folder selection situated in the top of MATLAB window interface
- Using the terminal command "cd path_to_MPTK/mptk/matlab"

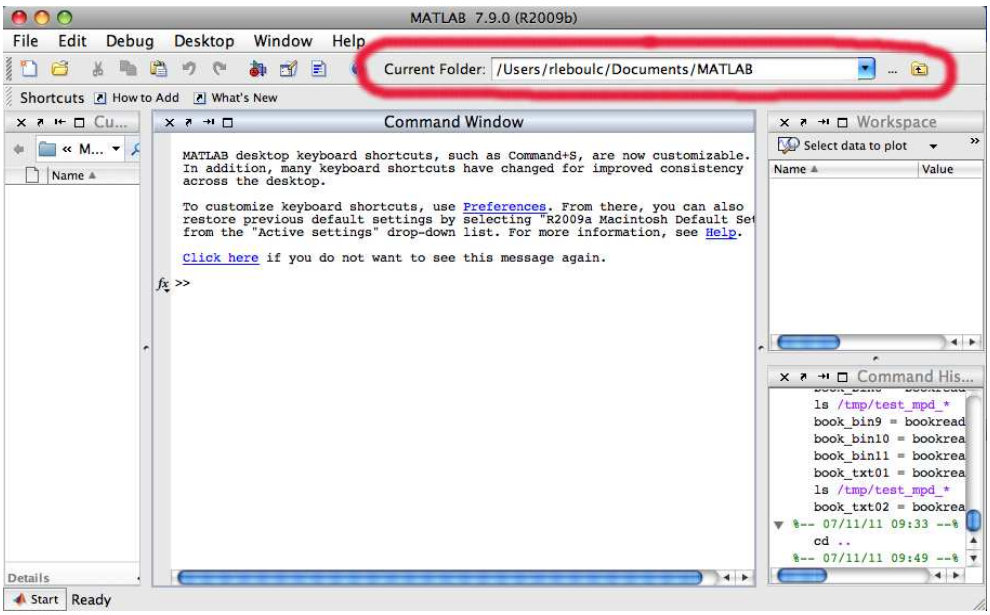

Figure 13.1: Description of Matlab selection path

13.2 Understanding Matlab MEX commands

The second step consist understanding how the Matlab MEX files works. In fact, "MEX" stands for MATLAB Executable. MEX-files are dynamically linked subroutines produced from C, C++ or Fortran source code that, when compiled, can be run from within MATLAB in the same way as MATLAB files or built-in functions.

MPTK provides Matlab MEX files and the extension of these files depend on the architecture of the user system. If Matlab is installed under your computer, follow these instructions in order to make Matlab MEX files work.

1. Launch Mathworks Matlab
2. Select the current folder as "path_to_MPTK/mptk/matlab". You should see the MEX-files related to your computer, depending on their extension :

   - Mexw32 for Windows 32 bytes - Mexw64 for Windows 64 bytes
   - Mexmaci for Mac 32 bytes - Mexmaci64 for Mac 64 bytes
   - Mexglx for Linux 32 bytes - Mexa64 for Linux 64 bytes

3. Run GettingStarted in order to test MPTK

13.3 Matlab book structure definition

The book structure has been fully rewritten to fit "bookedit" needs and to allow fast operations on atoms. This structure is extensively used in "GettingStarted.m" so you can find many examples of processing a book with matlab.

The original idea for the organization of the book is that atoms with the same sets of parameters (atoms of the same type: gabor, dirac, mdct, constant, ...) are gathered in atom sub structures for fast group processing.

In order to further ease common atom manipulations, this idea has evolved into also grouping atoms of the same lengths. Now, in the book structure, there are as many "atom" as [atoms type*length]. This leads to "book.atom" structures rather close to the different "blocks" in the MP_dictionary definition.

The 'book' structure contains the following fields:

- numAtoms : Number of atoms actually stored in the book
- numChans : number of channels of the atoms
- numSamples : number of samples in each channel covered by the reconstructed book
- sampleRate : signal sample rate
- maxNumAtoms : Number of atoms that could fit in the storage space
- atom: [1 x Ntype struct]
  - type : string
  - params: structure of atoms parameters arrays, whose field are type dependent

14. List of Matlab functions

Some Matlab utilities are bundled with the distribution to help loading and visualizing books under Matlab. They are automatically installed and can be found under mptk installation directory

The stable included files are:

- GettingStarted: A getting started example to use MPTK
- bookread: to load a binary book into Matlab
- bookwrite: to save a structured book into binary or txt format
- dictread: to read a dicitonary
- dictwrite: to write a dictionary
- mpdecomp: to decompose a signal into a book
- mprecons: to reconstruct a signal from a book
- sigread: reads and inport into matlab format an audio signal
- sigwrite: exports an audio signal from matlab to wav format

The non-stable included files are:

- bookedit: to manually edit any book according to its structure
- bookplot: to plot a book in a spectrogram-like fashion
- bookover: to overlay a book plot on a STFT spectrogram

14.1 Getting Started

```
Usage :
      info = getmptkinfo()

  Synopsis :
      Getting started example under Matlab to help using MPTK

  Output :
      The structure info has the following informations
          * info.atoms(x,1), x is the position of the atom
              - type              The type of the atom (Anywave, Dirac...)
          * info.blocks(x,1).parameters, x is the position of the block
              - name            The name of the block parameter (windowShift, blockOffset...)
              - type            The type of the block parameter (ulong, real...)
              - info            Details about the block parameter
              - default         Details about the block parameter
          * info.windows(x,1), x is the position of the info
              - type            The name of the window (hamming, flattop...)
              - needsOption     A boolean describing if otions are needed
          * info.path Includes different path names used by MPTK
```

14.2 bookread

```
Usage :
    book = bookread(filename)

  Synopsis:
    Imports a Matching Pursuit book file to Matlab, using MPTK and returns a structured book.
    The file can be either in binary mode (filename.bin) or in text mode (filename.xml)

  Input :
    * filename : The filename where to read the book

  Output:
    * book     : A book structure
```

14.3 bookwrite

```
Usage :
    bookwrite(book,filename[,writeMode])

  Synopsis:
    Exports a binary Matching Pursuit book file from Matlab, writes it under filename
    The exported file can be in binary format (filename.bin) or text format (filename.xml)

  Input :
    * book      : A book structure
    * filename  : The filename where to read the book
    * writeMode : OPTIONAL The book write mode ('binary' by default or 'txt')
```

14.4 bookedit

```
Usage :
    bookedit
    bookedit(mptkBook.bin)
    bookedit(book,chan)

  Synopsis :
    Interface for plotting and editing a Matching Pursuit book

  Detailed description :
    * bookedit asks the user for a MPTK binary book file and plots it. You can set a .mat
      variable called MPTKdir.mat containing the full path to your book directory:
          bookdir = /pathToMptk/book/;
          save MPTKdir.mat bookdir
    * bookedit(mptkBook.bin) loads the bookmptkbook.bin
    * bookedit(book,chan) reads the channel number chan of a book structure in the
        current axes. If it is a string, it is understood as a filename and the book is read
        from the corresponding file. Books can be read separately using the bookread utility.

  Notes :
    The patches delimit the support of the atoms. Their color is proportional to the atoms
  amplitudes, mapped to the current colormap and the current caxis.
```

14.5 bookplot

```
Usage :
    bookplot(book)
    bookplot(book,chan)
    bookplot(book,chan,bwfactor)

  Synopsis :
    Plots a Matching Pursuit book in the current axes

  Detailed description :
    * bookplot(book) plots the default channel (first channel).
    * bookplot(book,chan) plots the channel number chan of a MPTK book structure in
      the current axes. If book is a string, it is understood as a filename and the
      book is read from the corresponding file. Books can be read separately using
      the bookread utility.
    * bookplot(book,chan,bwfactor) allows to specify the bandwidths of the atoms,
      calculated as: bw = ( fs / (atom.length(channel)/2) ) / bwfactor; where fs is
      the signal sample frequency. When omitted, bwfactor defaults to 2.

  Notes :
     The patches delimit the support of the atoms. Their color is proportional to the atoms
  amplitudes, mapped to the current colormap and the current axis.
```

14.6 bookover

```
Usage :
    bookover(book,sig)
    bookover(book,sig,chan)

  Synopsis :
    Overlays a book plot on a STFT spectrogram

  Detailed description :
    * bookover(book,sig) plots the given book over a STFT spectrogram of the given signal
      sig using the default channel (channel 1).
    * bookover(book,sig,chan) plots the given book over a STFT spectrogram of the given
      signal sig, for channel number chan. The book and/or the signal can be given as
      filenames (WAV format for the signal).

  Notes :
     bookover will resize the current axes to fit the spectrogram area.
     The patches indicate the locations of the atom supports. The color indicates the energy
  level according to the JET colormap (more energy --> closer to red, less energy --> closer to blue)
```

14.7 dictread

```
Usage :
    dict = dictread(filename)

  Synopsis :
    Imports a dictionary description dict from a file filename to Matlab, using MPTK

  Input :
    * filename : the filename where to read the dictionary

  Output :
    * dict : a dictionary description

  Detailed description :
```

```
  dict is a dictionary description with the following structure dict.blocki = block
where, for example block.type = dirac and block may have other field names
```

14.8 dictwrite

```
Usage:
    isvalid = dictwrite(dict[,filename])

  Synopsis :
    Exports a dictionary description dict from Matlab to a file filename, using MPTK. It
  returns isvalid, a boolean describing if the syntax of dict is correctly formed

  Input :
    * dict : a dictionary description
    * filename : OPTIONAL the filename where to read the dictionary

  Output :
     isvalid : indicates if the dictionary structure was correctly formed.

  Detailed description :
    dict is a dictionary description with the following structure dict.blocki = block
  where, for example block.type = dirac and block may have other field names
```

14.9 mpdecomp

```
Usage :
    [book,residual,decay] = mpdecomp(signal,sampleRate,dict,numIter)

  Synopsis :
    Decompose a signal signal using its sampling frequency sampleRate, a dictionary
  structure dict performing numIter iterations and gives the resulting decomposition
  book, the residual obtained after the iterations and decay, a vector with the
  energy of the residual after each iteration

  Inputs:
    signal    : a numSamples x numChans signal (each column is a channel)
    sampleRate: the sampleRate of the signal
    dict      : either a dictionary Matlab strucure, or a filename
    numIter   : the number of iterations to perform

  Outputs:
    book      : the book with the resulting decomposition (warning : its sampleRate is 1 by default)
    residual  : the residual obtained after the iterations
    decay     : a numIter x 1 vector with the energy of the residual after each iteration
```

14.10 mprecons

```
Usage :
    signal = mprecons(book[,residual])

  Synopsis :
    Reconstruct a signal signal from a book. An optional residual can be added,
    under the condition that its dimension matches the fields book.numSamples
    and book.numChans.

  Inputs :
    book     : a book Matlab structure
    residual : OPTIONAL a numSamples x numChans matrix, which dimensions should
               match the fields book.numSamples, book.numChans

  Outputs :
    signal   : the reconstructed signal, a numSamples x numChans matrix
```

14.11 sigread

```
Usage :
    [signal,sampleRate] = sigread(filename)

  Synopsis:
    Reads an imports signal filename of any format supported by libsndfile library
    to Matlab and gives a matrix signal (numSamples x numChans) and the sampling
    frequency of the read signal sampleRate.

  Input :
    filename   : The filename where to read the signal

  Output:
    signal     : A matrix numSamples x numChans
    sampleRate : The sampling frequency of the read signal
```

14.12 sigwrite

```
Usage :
    sigwrite(signal,fileName,sampleRate)

  Synopsis:
    Exports a signal of any format supported by libsndfile library from Matlab using
    the sampling frequency sampleRate and writes it under filename.

  Inputs :
    signal    : A numSamples x numChans matrix
    filename  : The filename where to write the signal
    sampleRate: The sampling frequency
```

15. Matlab bookedit GUI

This GUI makes intensive use of function_handles (denoted @functionName). The differents callbacks of the GUI are function handles which refer to sub functions. As a consequence, the code is divided in a large number of subfunctions. A tip to navigate inside the code is to select the function name, right-click and "open selection" or F4 to jump to the sub-function.

The data (book info, selection info, figure handles and other variables) are organised in structure and are stored inside the figure. The data structure is loaded at the beginning of many functions and saved back at their end using the commands:

- data = get(gcbf,'UserData'); % load data
- set(gcbf,'UserData',data); % save data

15.1 Interface startup

The interface can be started under Matlab with bookedit command with different arguments:

- No argument : At startup, a find file window asks the user for a valid MPTK book
- bookname: a string with a valid MPTK book. In this case the book is loaded at startup

15.2 Overview and main functionalities

Bookedit is a matlab tool to edit MTPK books issued from decomposition. It implements plotting tools, basic editing functions and audio transformations:

- Plot atoms, toggle view per type and length
- Select atoms, Move, Cut or apply a gain on selected atoms
- Pitch Shifting and Time stretching of atoms
- Load / Save the modified MPTK book in native MPTK format
- Reconstruct and play the transformed/selected book

The following figure shows a snapshot of the interface:

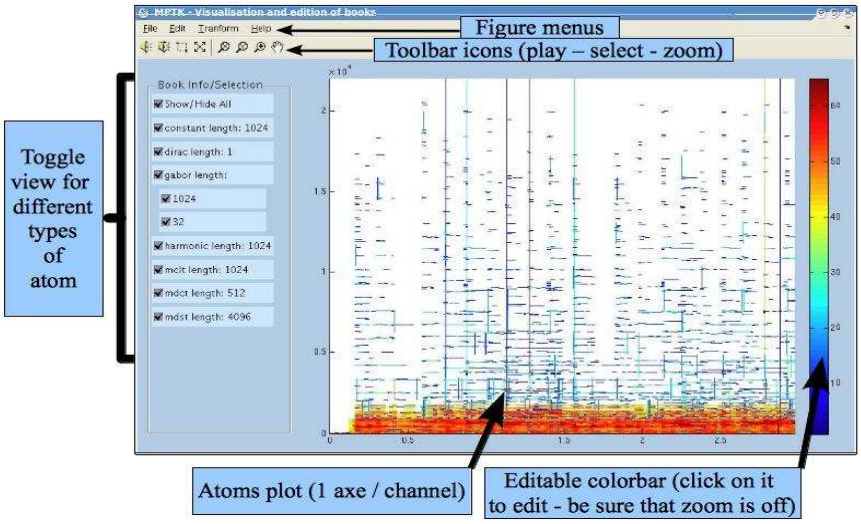

Figure 15.1: bookedit main page description

15.3 Toolbar functions

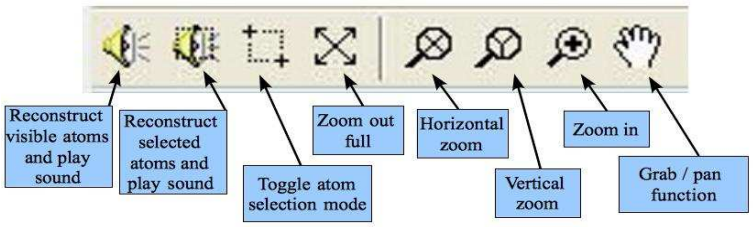

Figure 15.2: bookedit file toolbar description

Atom selection: When the atom selection is toggled on, you can directly drag a rectangle in the plot to select an area.
Important note: when a zoom function is active (icon grey) the colormap cannot be edited.

15.4 File Menu functions

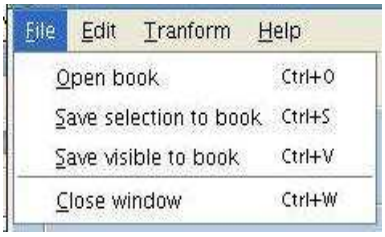

Figure 15.3: bookedit file toolbar description

The ‘Edit’ menu contains functions for editing the selected atoms. It also has a ‘Refresh figure’ usefull in case the GUI gets lost.

15.4.1 Edit Menu functions

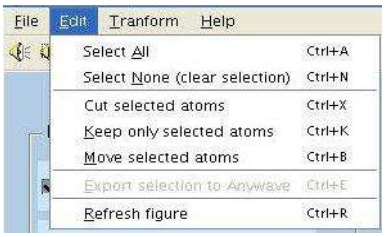

Figure 15.4: bookedit edit toolbar description

The ‘File’ menu contains functions for loading and saving a book. It possible to either save only the selected atoms or the visible part of the book.

15.4.2 Transform Menu functions

Figure 15.5: bookedit transform toolbar description

The ‘Transform’ menu contains functions for typical processing of audio signals. Some functions are not implemented and thus not accessible.

16. Testing if everything works

If you experiment some issues during the usage of MPTK, if you would like to ensure that the executables are working correctly, if you want to try new dictionaries, a series of tests are available on the source release.

16.1 Local tests using Ctest

CTest is a testing tool distributed as a part of CMake. It can be used to automate updating, configuring, building, testing, performing memory checking, performing coverage, and submitting results to a CDash or Dart dashboard system.

You can run local tests on your machine using the command "ctest" if :

- MPTK has been successfully pre-compiled with cmake
- MPTK has been successfully compiled
- MPTK has been successfully installed

```
pachamama:bar rleboulc$ ctest
Test project /Users/rleboulc/bar
      Start  1: HELP_MPD,
 1/116 Test  #1: HELP_MPD, ..............................  Passed    0.27 sec
      Start  2: HELP_GPD,
 2/116 Test  #2: HELP_GPD, ..............................  Passed    0.02 sec
      Start  3: HELP_MPR,
 3/116 Test  #3: HELP_MPR, ..............................  Passed    0.01 sec
      Start  4: HELP_MPF,
 4/116 Test  #4: HELP_MPF, ..............................  Passed    0.01 sec
      Start  5: HELP_MPCAT,
 5/116 Test  #5: HELP_MPCAT, ............................  Passed    0.00 sec
      Start  6: HELP_MPVIEW,
 6/116 Test  #6: HELP_MPVIEW, ...........................  Passed    0.02 sec
      Start  7: HELP_MPDDEMIX,
 7/116 Test  #7: HELP_MPDDEMIX, .........................  Passed    0.01 sec
      Start  8: SCRATCH_MPD_RELATIVE,
 8/116 Test  #8: SCRATCH_MPD_RELATIVE, ..................  Passed    1.62 sec
      Start  9: SCRATCH_MPD_REPORT,
```

Figure 16.1: cTtest testing report description

17. Format of the dictionary files

The dictionary files use a XML syntax, with tags enclosed between angle brackets. Strict XML compliance is now mandatory. [1].

17.1 General rules

A dictionary should include, in the following order:

1. An optional XML declaration line;
2. A mandatory dictionary opening tag;
3. An optional library version tag;
4. A list of blocks with their parameters (some mandatory, some admitting default values);
5. A dictionary closing tag.

Any text included after the dictionary closing tag will be ignored. Blank spaces and line breaks are ignored. Any part of the file can be commented out either by enclosing it between `<!--` and `-->` (XML style) The parser will send any text it can't match to stderr with a error message: in the event of syntax errors in a dictionary, some dictionary pieces will therefore show up in stderr.

Valid tags

<?xml version="1.0" encoding="ISO-8859-1"?>
: The XML declaration line.

<dict>List of blocks</dict>
: The opening and closing tags for the dictionary. Anything coming after the </dict> tag will be ignored by the parser.

<libVersion>blah</libVersion> [optional]
: The library version declaration. This is provided for backward compatibility if we ever change the dictionary syntax. If absent, the library version is taken to be the current one.

<blockproperties name="PROPERTIES_NAME"> List of block parameters</blockproperties>
: The opening and closing tags for a block properties list.

<block uses="PROPERTIES_NAME"> List of block parameters</block>
: The opening and closing tags for a block. The block will be construct using the list of parameters defined in PROPERTIES_NAME block properties.

<block>List of block parameters</block>
: The opening and closing tags for a block. the list of block parameters should contains all the mandatories parameters for this type of block

<param name="NAME" value="VALUE"/>
: A block parameter. The NAME and corresponding VALUE value in string. The parameter value is set to the corresponding type when the dictionary file is parsed. Note that when using the <block uses="PROPERTIES_NAME">List of block parameters </block> syntax. The block parameters defined in the block list override the parameters from PROPERTIES_NAME block properties.

[1] Uh, that is, if a DTD get ready someday.

The parameters list for block types (**G**abor, **H**armonic, **C**hirp, **A**nywave, **C**onsTant, **N**yquist):

- `windowLen (G,H,C,CT,N)`
  Unsigned long int value: The atom length (i.e., the window length in the STFT)
- `windowShift (G,H,C,A,CT,N)`
  Unsigned long int value: The atom shift (i.e., the window shift in the STFT)
- `windowRate (G,H,C)`
  Double value between 0 and 1: An other way to specify the window shift, as a proportion of the `windowLen`. If both are present, `windowShift` has precedence over `windowRate`
- `fftSize (G,H,C)`
  Unsigned long value: The frequency resolution in terms of FFT size. It has to be greater or equal to `windowLen` if `windowLen` is even, or $\geq$`(windowLen+1)` if `windowLen` is odd. If absent, defaults to `windowLen` (or `windowLen+1`).
- `blockOffset (G,H,C,CT,N)`
  Unsigned long value: the block Offset, (i.e. the position of the first frame If absent, defaults to 0
- `windowtype (G,H,C,CT,N)`
  The window specification to be included among the block parameters
- `windowopt (G,H,C,CT,N)`
  The optional parameter for window type parameter :
  - Used for: hamgen, gauss, exponential
  - Not used for: rectangle, triangle, cosine, hanning, hamming, blackman, flattop, fof

  The meaning of the optional parameter varies according to the window type
- `f0Min (H)`
  Double value: minimum frequency (in Hz) from which the fundamental frequency of the harmonic atoms is searched. Defaults to the first non-null FFT frequency
- `f0Max (H)`
  Double value: maximum frequency (in Hz) at which the fundamental frequency of the harmonic atoms is searched. Defaults to the Nyquist frequency of the considered signal
- `numPartials (H)`
  Unsigned int value: number of partials considered when tracking the harmonic atoms
- `numFitPoints (C)`
  (EXPERIMENTAL) Unsigned int value: number of polynomial fitting points considered for the chirp optimization algorithm. Defaults to 1.
- `numIter (C)`
  (EXPERIMENTAL) Unsigned int value: number of iterations considered for the chirp optimization algorithm. Defaults to 1.
- `tableFileName (A)`
  String: filename of the table containing the waveforms (ex: /udd/toto/table.bin). Note that there is no " around the string.

Note : The `dirac` blocks don't need any parameter (they just match signal samples).

Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>                    :The optional XML declaration line.
<dict>                                                         :The dictionary opening tag.
     <libVersion>0.2</libVersion>                          :The optional library version.
     <blockproperties name="GAUSS-WINDOW">                 :A new block properties.
               <param name="windowtype" value="gauss"/>
               <param name="windowopt" value="0.02"/>      :Gauss windows need a parameter.
     </blockproperties>
     <block uses="GAUSS-WINDOW">                           :New block using GAUSS-WINDOW.
               <param name="type" value="gabor"/>
               <param name="windowLen" value="32"/>
               <param name="windowShift" value="32"/>
               <param name="fftSize" value="32"/>
     </block>
     <block>
               <param name="type" value="anywave"/>        :An anywave block.
               <param name="tableFileName" value="/udd/toto/table.xml"/>
               <param name="windowShift" value="1"/>
     </block>
     <block>
               <param name="type" value="dirac"/>          :A dirac block.
     </block>
     <block>
               <param name="type" value="constant"/>       :A constant block.
               <param name="windowLen" value="512"/>
               <param name="windowShift" value="32"/>
     </block>
     <block>
               <param name="type" value="nyquist"/>        :A nyquist block.
               <param name="windowLen" value="512"/>
               <param name="windowShift" value="32"/>
     </block>
</dict>
```

17.2 Anywave table

Waveforms need have been loaded before using anywave atoms. That's the difference between parametric atoms, such as Dirac, Gabor, ... and anywave atoms. Therefore, a different syntax has to be employed. In the dictionary file, the anywave block points to a "anywave table definition file" (/udd/toto/table.bin in the example). This file has a XML syntax, to give all the parameters of the waveforms, and points to a binary "anywave table data file", that gives the data (/udd/toto/table_data.bin in the example).

Note that one table corresponds to one atom length, and one number of channels. To use several atom lengths, create several anywave tables, and define several "anywave" blocks.

Valid tags

<?xml version="1.0" encoding="ISO-8859-1"?>
The XML declaration line.

<dict>List of blocks</dict>
The opening and closing tags for the dictionary. Anything coming after the </dict> tag will be ignored by the parser.

<libVersion>blah</libVersion> [optional]
The library version declaration. This is provided for backward compatibility if we ever change the dictionary syntax. If absent, the library version is taken to be the current one.

<par type="NAME" value="VALUE"/>
A table parameter. The NAME and corresponding VALUE value in string. The parameter value is set to the corresponding type when the table file is parsed. The NAME and corresponding VALUE types are given below:

- numChans (Unsigned short int): The number of channels of the atoms
- filterLen (Unsigned long int): The length of the anywave waveforms in the table
- numFilters (Unsigned long int): The number of anywave waveforms in the table
- data (String): Name of the file containing the waveforms

The "anywave table data file" is binary, and built as follows: all the waveforms are written in double type, one after the other, and for each, one channel after the other. The following scheme is for two channels and four waveforms:

—w1c1—w1c2—w2c1—w2c2—w3c1—w3c2—w4c1—w4c2—

Example of an anywave table

```
<?xml version="1.0" encoding="ISO-8859-1" ?>                    :The optional XML declaration line.
<dict>
     <libVersion>0.6.0</libVersion>                          :The optional library version.
     <block>
          <param name="blockOffset" value="" />
          <param name="data" value="+HxVeGKDh7946h9..." /> :The binary datas encoded with base64 (text format)
          <param name="filterLen" value="128" />
          <param name="numChans" value="1" />
          <param name="numFilters" value="4" />
          <param name="windowShift" value="1" />
     </block>
</dict>
```

17.3 MDCT/MDST/MCLT

Definition

The Modified Discrete Cosine Transform (MDCT) and the Modified Discrete Sine Transform (MDST) are two orthogonal transforms based on local cosine functions. The Modulated Complex Lapped Transform (MCLT) is the complex extension such as $MCLT = MDCT + i * MDST$. The atoms corresponding to the MCLT of a signal of length $N = PL$ and a window length of $2L$, are defined as:

$$x_{p,k}(n) = w(n - pL)exp\left[i\frac{\pi}{L}\left((n - pL) + \frac{1}{2} + \frac{L}{2}\right)\left(k + \frac{1}{2}\right)\right] \tag{17.1}$$

with $n = 0, .., N-1$, $k = 0, .., L-1$ and $p = 0, .., P-1$. w is a window which is complementary in energy i.e. it verifies $w^2(n) + w^2(n + L) = 1, n = 0, .., L - 1$.

The atoms corresponding to the MDCT are defined as the real part of the previous formula and the atoms corresponding to the MDST the imaginary part of the previous formula. We also define a generalized MDCT/MDST/MCLT where the window shift, the window shape and the fft size are not constraint by the orthogonality property. The corresponding atoms for a signal of length $N = PS$, a window length of $2L$, a window shift of S, a fft size of $Nfft$ and a window w, are defined as:

$$x_{p,k}(n) = w(n - pS)exp\left[i\frac{2\pi}{Nfft}\left((n - pS) + \frac{1}{2} + \frac{L}{2}\right)\left(k + \frac{1}{2}\right)\right] \text{ if } Nfft = 2L \tag{17.2}$$

$$x_{p,k}(n) = w(n - pS)exp\left[i\frac{2\pi}{Nfft}\left((n - pS) + \frac{1}{2} + \frac{L}{2}\right)(k)\right] \text{ if } Nfft = 2mL \tag{17.3}$$

with $n = 0, .., N - 1$, $k = 0, .., Nfft/2 - 1$, $p = 0, .., P - 1$ and m is even with $m >= 2$.

Valid tags

<?xml version="1.0" encoding="ISO-8859-1"?> [optional]
The XML declaration line.
<libVersion>blah</libVersion> [optional]
The library version declaration. This is provided for backward compatibility if we ever change the dictionary syntax. If absent, the library version is taken to be the current one.
<dict>List of blocks</dict>
The opening and closing tags for the dictionary. Anything coming after the </dict> tag will be ignored by the parser.
<par type="NAME" value="VALUE"/>
A table parameter. The NAME and corresponding VALUE value in string. The parameter

```
value is set to the corresponding type when the table parameter file is parsed. The NAME
and corresponding VALUE types are given below:
     - windowLen (Unsigned long int): The atom length (window length). It has to be even
     - windowShift (Unsigned long int): The atom shift (window shift)
     - windowRate (Double between 0.0 and 1.0:): An alternate way to spe-cify the window
       shift, as a proportion of the windowLen. If both are present, windowShift has precedence
       over windowRate
     - fftSize (Unsigned long): The frequency resolution in terms of FFT size. It has to be equal
       to windowLen or a multiple of 2*windowLen
     - blockOffset (Unsigned long): The block Offset (position of the first frame). If absent, defaults to 0
     - windowtype (Unsigned char): The window specification to be included among the block parameters
     - windowopt (Double): The optional parameter for window type:
         * Used: hamgen, gauss, exponential, kbd
         * Not used: rectangle, triangle, cosine, hanning, hamming, blackman, flattop, fof
```

A default MDCT/MDST/MCLT block is defined using 2 parameters, the window length and the window type. The window type must be `rectangle`, `cosine` or `kbd`.

An example of a MDCT dictionnary is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
        <dict>
        <libVersion>0.2</libVersion>
                <block>
                        <param name="type" value="mdct"/>
                        <param name="windowLen" value="2048"/>
                        <param name="windowtype" value="kbd"/>
                        <param name="windowopt" value="5"/>
                </block>
        </dict>
```

An example of a generalized MCLT dictionnary is:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
        <dict>
        <libVersion>0.2</libVersion>
                <block>
                        <param name="type" value="mdct"/>
                        <param name="windowLen" value="2048"/>
                        <param name="fftSize" value="4096"/>
                        <param name="windowtype" value="cosine"/>
                        <param name="windowopt" value="0"/>
                </block>
        </dict>
```

18. Format of the book files

The book files can be used either as a binary or XML files. They are divided into two different parts. The first part (xml style) includes all the different dictionaries used during the decomposition. The second part (xml or binary style) represents the differents atom founded during the decomposition.

18.1 General rules

A book should include, in the following order:

1. An optional XML declaration line;
2. A dictionary XML structure (see section 17.1)
3. A 'bin' or 'txt' string refering to the following book structure
4. A book opening tag (including mandatory parameters)
5. A book closing tag.

Any text included after the book closing tag will be ignored. Blank spaces and line breaks are ignored. Any part of the file can be commented out either by enclosing it between `<!--` and `-->` (XML style) The parser will send any text it can't match to stderr with a error message: in the event of syntax errors in a dictionary, some dictionary pieces will therefore show up in stderr.

Valid tags

<?xml version="1.0" encoding="ISO-8859-1"?>
: The XML declaration line.

<dict>List of blocks</dict>
: The opening and closing tags for the dictionary. Anything coming after the </dict> tag will be ignored by the parser.

<libVersion>blah</libVersion> [optional]
: The library version declaration. This is provided for backward compatibility if we ever change the dictionary syntax. If absent, the library version is taken to be the current one.

<blockproperties name="PROPERTIES_NAME"> List of block parameters</blockproperties>
: The opening and closing tags for a block properties list.

<block uses="PROPERTIES_NAME"> List of block parameters</block>
: The opening and closing tags for a block. The block will be construct using the list of parameters defined in PROPERTIES_NAME block properties.

<block>List of block parameters</block>
: The opening and closing tags for a block. the list of block parameters should contains all the mandatories parameters for this type of block

<param name="NAME" value="VALUE"/>
: A block parameter. The NAME and corresponding VALUE value in string. The parameter value is set to the corresponding type when the dictionary file is parsed. Note that when using the <block uses="PROPERTIES_NAME">List of block parameters </block> syntax. The block parameters defined in the block list override the parameters from PROPERTIES_NAME block properties.

'bin' or 'txt' string
: A string describing if the following book structure is in binary format or in text (XML) format

<book nAtom=''x'' numChans=''x'' numSamples=''x'' sampleRate="x" libVersion="x.x.x">
: The book opening tag. It includes the number of atoms founded, the number of channels of the signal, the number of samples and the sample rate asked by the user and the library version

xwA&=<?#_8zk?gabor*
: An atom datas. It includes the binary datas (''xwA...'') and the name of the type of the atom

</book>

The book closing tag.

A. Getting libsndfile

Libsndfile is a C library for reading and writing files containing sampled sound (such as MS Windows WAV and the Apple/SGI AIFF format) through one standard library interface. It is released in source code format under the Gnu Lesser General Public License.

This library can be downloaded either through the web (for example via Mega-Nerd website) or in the "relased files" section of the MPTK Gforge:

- http://www.mega-nerd.com/libsndfile/
- http://mptk.gforge.inria.fr/

Install this package on your computer using the instructions given on this web site will be easy an will yield the following sequence of commands after unzipping the package:

```
./configure [options]
make
make install
```

B. Getting fftw

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST).

This library can be downloaded either through the web or in the "relased file" section of the MPTK Gforge:

- http://www.fftw.org/install/windows.html
- http://mptk.gforge.inria.fr/

Install this package on your computer using the instructions given on this web site will be easy an will yield the following sequence of commands after unzipping the package:
./configure [options]
make
make install

C. References

C.1 Algorithmic aspects

For more information about the algorithmic aspects of the Matching Pursuit algorithm, see:
[Zhang, 1993], [Mallat and Zhang, 1993], [Pati et al., 1993], [Davis, 1994], [Bergeaud, 1995], [Bergeaud and Mallat, 1996], [Jaggi et al., 1998], [Gribonval et al., 1996a], [Gribonval et al., 1996b], [Davis et al., 1997], [Goodwin and Vetterli, 1999], [Gribonval, 1999], [Gribonval, 2001b], [Gribonval, 2002], [Gribonval and Bacry, 2003], [Gribonval, 2003], [Krstulović and Gribonval, 2006].

C.2 Theoretic aspects

For more information about the theoretic aspects of the Matching Pursuit algorithm, see:
[Gribonval, 2001a], [Gribonval and Nielsen, 2001], [Gribonval, 2001c], [Gribonval and Nielsen, 2003b], [Gribonval and Nielsen, 2003c], [Gribonval and Vandergheynst, 2004], [Gribonval and Nielsen, 2003a], [Gribonval et al., 2004].

C.3 Experimental results and applications

Examples of experimental and applicative results obtained with MPTK can be found in:
[Krstulović et al., 2005], [Lesage et al., 2006].

Bibliography

[Bergeaud, 1995] Bergeaud, F. (1995). *Représentations adaptatives d'images numériques, Matching Pursuit*. PhD thesis, Ecole Centrale Paris.

[Bergeaud and Mallat, 1996] Bergeaud, F. and Mallat, S. (1996). Matching pursuit : Adaptive representations of images and sounds. *Computational and Applied Mathematics*, 15(2):97–109.

[Davis, 1994] Davis, G. (1994). *Adaptive Nonlinear Approximations*. PhD thesis, New York University.

[Davis et al., 1997] Davis, G., Mallat, S., and Avellaneda, M. (1997). Adaptive greedy approximations. *Constr. Approx.*, 13(1):57–98.

[Goodwin and Vetterli, 1999] Goodwin, M. and Vetterli, M. (1999). Matching pursuit and atomic signal models based on recursive filter banks. *IEEE Trans. on Signal Proc.*, 47(7):1890–1902.

[Gribonval, 1999] Gribonval, R. (1999). *Approximations non-linéaires pour l'analyse de signaux sonores*. PhD thesis, Université Paris IX Dauphine.

[Gribonval, 2001a] Gribonval, R. (2001a). A counter-example to the general convergence of partially greedy algorithms. *Journal of Adaptive Theory*, 111:128–138. doi:10.1006/jath.2001.3556.

[Gribonval, 2001b] Gribonval, R. (2001b). Fast matching pursuit with a multiscale dictionary of Gaussian chirps. *IEEE Trans. Sig. Proc.*, 49(5):994–1001.

[Gribonval, 2001c] Gribonval, R. (2001c). Partially greedy algorithms. In Kopotun, K., Lyche, T., and Neamtu, M., editors, *Trends in Approximation Theory*, pages 143–148, Nashville, TN. Vanderbilt University Press.

[Gribonval, 2002] Gribonval, R. (2002). Sparse decomposition of stereo signals with matching pursuit and application to blind separation of more than two sources from a stereo mixture. In *Proc. Int. Conf. Acoust. Speech Signal Process. (ICASSP'02)*, Orlando, Florida.

[Gribonval, 2003] Gribonval, R. (2003). Piecewise linear source separation. In Unser, M., Aldroubi, A., and Laine, A., editors, *Proc. SPIE '03*, volume 5207 Wavelets: Applications in Signal and Image Processing X, pages 297–310, San Diego, CA.

[Gribonval and Bacry, 2003] Gribonval, R. and Bacry, E. (2003). Harmonic decomposition of audio signals with matching pursuit. *IEEE Trans. Sig. Proc.*, 51(1):101–111.

[Gribonval et al., 1996a] Gribonval, R., Bacry, E., Mallat, S., Depalle, P., and Rodet, X. (1996a). Analysis of sound signals with high resolution matching pursuit. In *Proc. TFTS'96*, pages 125–128, Paris, France.

[Gribonval et al., 1996b] Gribonval, R., Depalle, P., Rodet, X., Bacry, E., and Mallat, S. (1996b). Sound signals decomposition using a high resolution matching pursuit. In *Proc. ICMC'96*, pages 293–296.

[Gribonval et al., 2004] Gribonval, R., Figueras i Ventura, R. M., and Vandergheynst, P. (2004). A simple test to check the optimality of sparse signal approximations. Technical report, IRISA, Rennes, France. submitted to EURASIP Signal Processing J.

[Gribonval and Nielsen, 2001] Gribonval, R. and Nielsen, M. (2001). Approximate weak greedy algorithms. *Advances in Computational Mathematics*, 14(4):361–378.

[Gribonval and Nielsen, 2003a] Gribonval, R. and Nielsen, M. (2003a). Highly sparse representations from dictionaries are unique and independent of the sparseness measure. Technical Report R-2003-16, Dept of Math. Sciences, Aalborg University.

[Gribonval and Nielsen, 2003b] Gribonval, R. and Nielsen, M. (2003b). Sparse decompositions in "incoherent" dictionaries. In *Proc. IEEE Intl. Conf. Image Proc. (ICIP'03)*, Barcelona, Spain.

[Gribonval and Nielsen, 2003c] Gribonval, R. and Nielsen, M. (2003c). Sparse decompositions in unions of bases. *IEEE Trans. Inform. Theory*, 49(12):3320–3325.

[Gribonval and Vandergheynst, 2004] Gribonval, R. and Vandergheynst, P. (2004). On the exponential convergence of Matching Pursuits in quasi-incoherent dictionaries. Technical Report PI-1619, IRISA, Rennes, France. submitted to IEEE Trans. Inf. Th.

[Jaggi et al., 1998] Jaggi, S., Carl, W., Mallat, S., and Willsky, A. (1998). High resolution pursuit for feature extraction. *Applied Computational Harmonic Analysis*, 5(4):428–449.

[Krstulović and Gribonval, 2006] Krstulović, S. and Gribonval, R. (2006). Matching pursuit made tractable. In *Proc.ICASSP 2006*.

[Krstulović et al., 2005] Krstulović, S., Gribonval, R., Leveau, P., and Daudet, L. (2005). A comparison of two extensions of the matching pursuit algorithm for the harmonic decomposition of sounds. In *Proc.WASPAA'05*.

[Lesage et al., 2006] Lesage, S., Krstulović, S., and Gribonval, R. (2006). Underdetermined source separation: comparison of two approaches based on sparse decompositions. In *Proc.ICA'06*.

[Mallat and Zhang, 1993] Mallat, S. and Zhang, Z. (1993). Matching pursuit with time-frequency dictionaries. *IEEE Trans. Sig. Proc.*, 41(12):3397–3415.

[Pati et al., 1993] Pati, Y., Rezaiifar, R., and Krishnaprasad, P. (1993). Orthonormal matching pursuit : recursive function approximation with applications to wavelet decomposition. In *Proceedings of the* 27^{th} *Annual Asilomar Conf. on Signals, Systems and Computers*.

[Zhang, 1993] Zhang, Z. (1993). *Matching Pursuit*. PhD thesis, New York University.